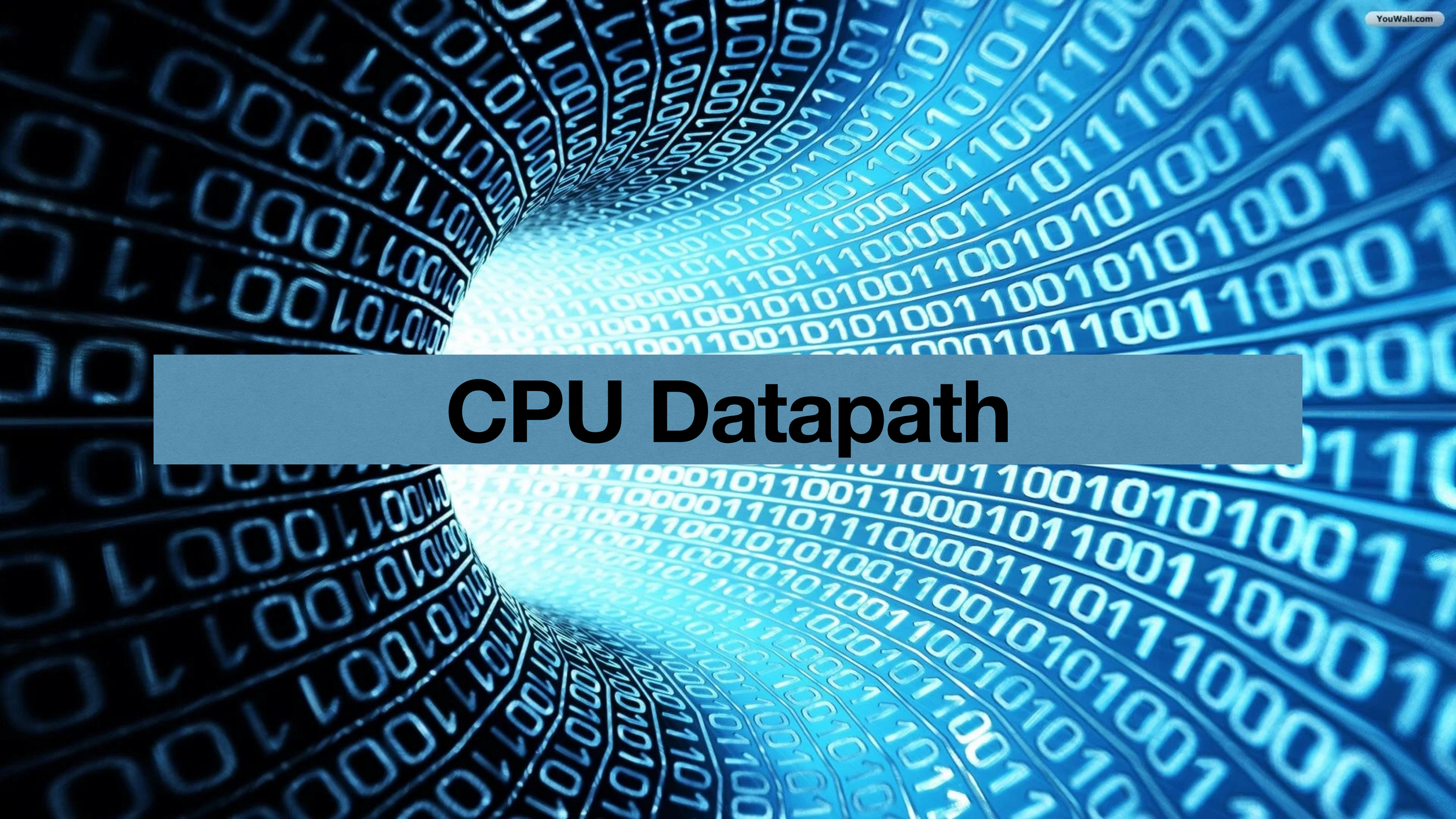
# Digital Logic Design + Computer Architecture

**Sayandeep Saha**

**Assistant Professor**
**Department of Computer Science and Engineering**
**Indian Institute of Technology Bombay**




# CPU Datapath

# What is “Datapath” for a Processor?

- Same as the datapath control split we studied for GCD processor.
  - **Datapath processes the data**
  - **Controller tells how to process the data**
- But here it’s for processing the instructions in an ISA
- A datapath that can process all the instructions in an ISA


Image generated by ChatGPT

# Datapath Elements

- We shall begin with the simplest case
  - Every instruction finishes in **one clock cycle**
  - The clock frequency is determined by the instruction finishes last
- Remember, again how we constructed the GCD datapath
- Let’s first identity what are the components needed

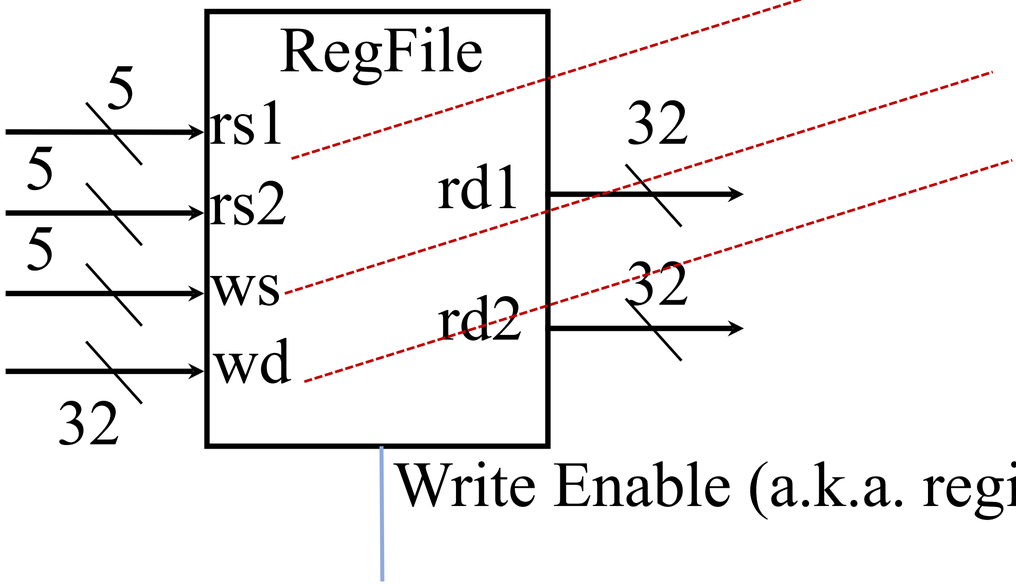

Image generated by ChatGPT

# Datapath Elements

- Remember, again how we constructed the GCD datapath
- Let’s first identity what are the components needed


Image generated by ChatGPT

# Datapath Elements: Register File

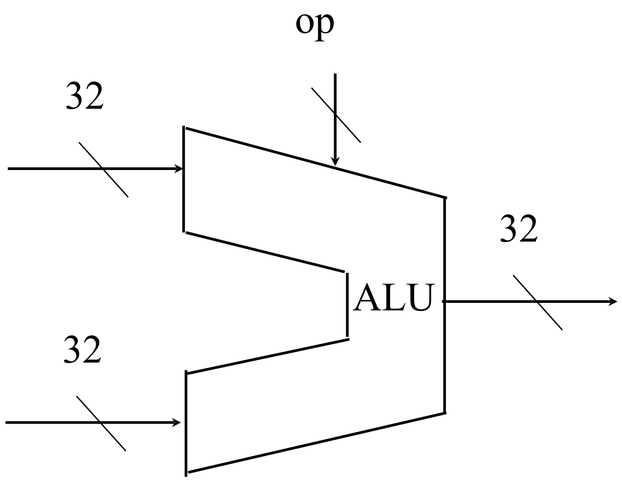



rs1 and rs2: Read register 1 and 2

ws: Write register

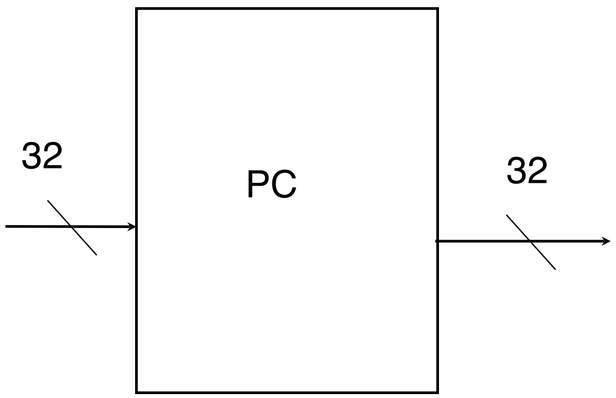
wd: Write data

# Datapath Elements: The ALU



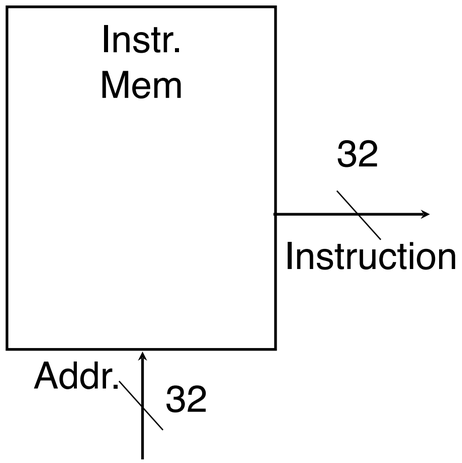

- All arithmetic and logical operation happens here
- Operation selection
- E.g.,
  - Op = 0: Y = A+B
  - OP = 1: Y = A- B
  - OP = 2; Y = A*B

# Datapath Elements: Program Counter




- Remember PC register??
- It always points to the next instruction to be executed…
- But where is the next instruction???

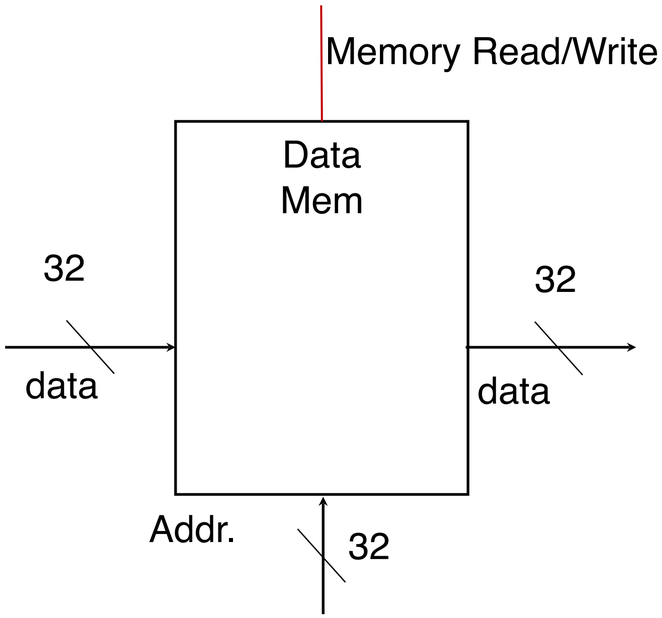
# Datapath Elements: Instruction Memory




Remember: No writes to instruction memory

Not concerned about how programs are loaded into this memory.

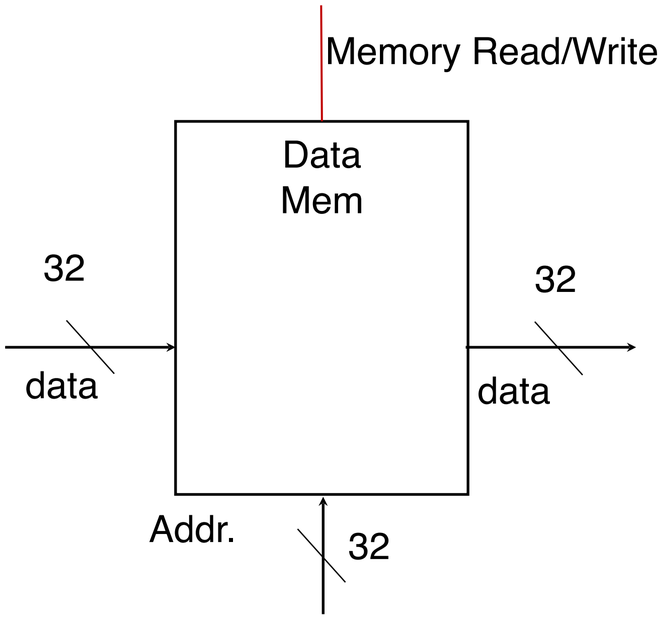
# Datapath Elements: Data Memory




Why data and instruction memory and not one memory?

- Recall Harvard vs. Von Neuman
- Not so simple matter, will discuss later.

# Datapath Elements: Data Memory




Why data and instruction memory and not one memory?

- Recall Harvard vs. Von Neuman
- Not so simple matter, will discuss later.

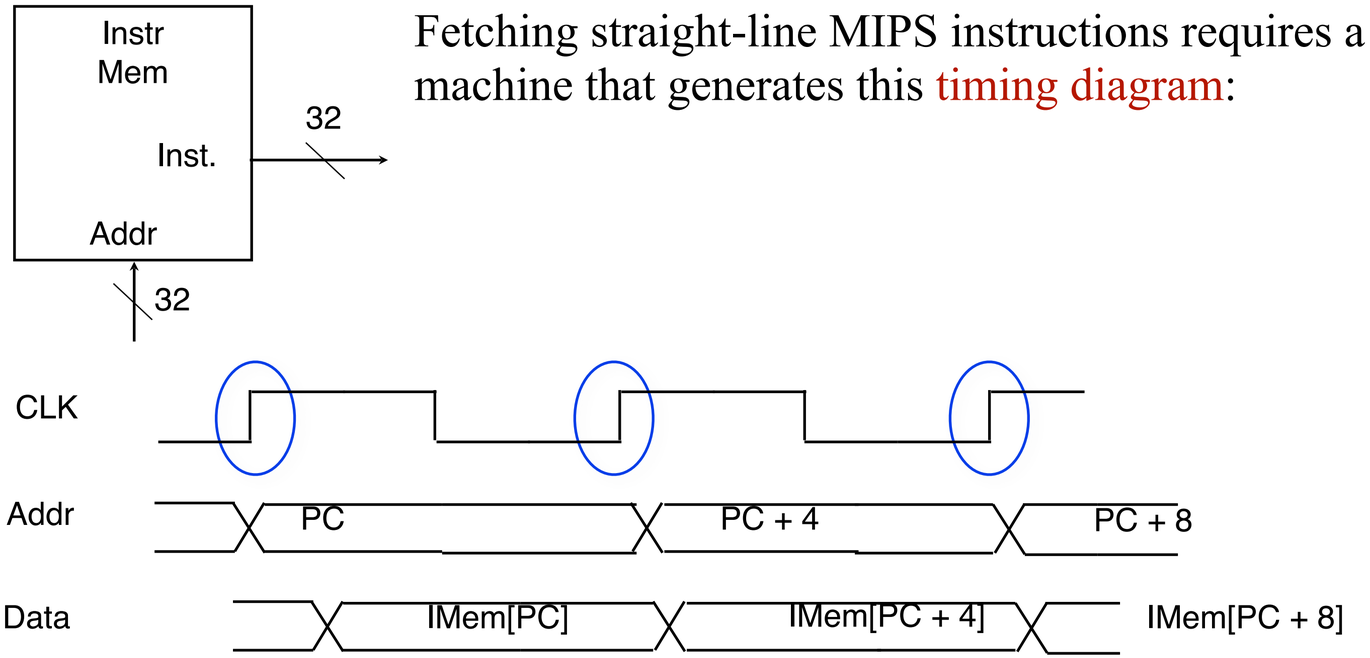
# Datapath Elements: Buses

- Same as your public transport
- Transfer data and instructions
- Separate buses for address and data

# Datapath Elements: Buses

- Same as your public transport
- Transfer data and instructions
- Separate buses for address and data
  - Why???

# Datapath Elements: Timing Diagram

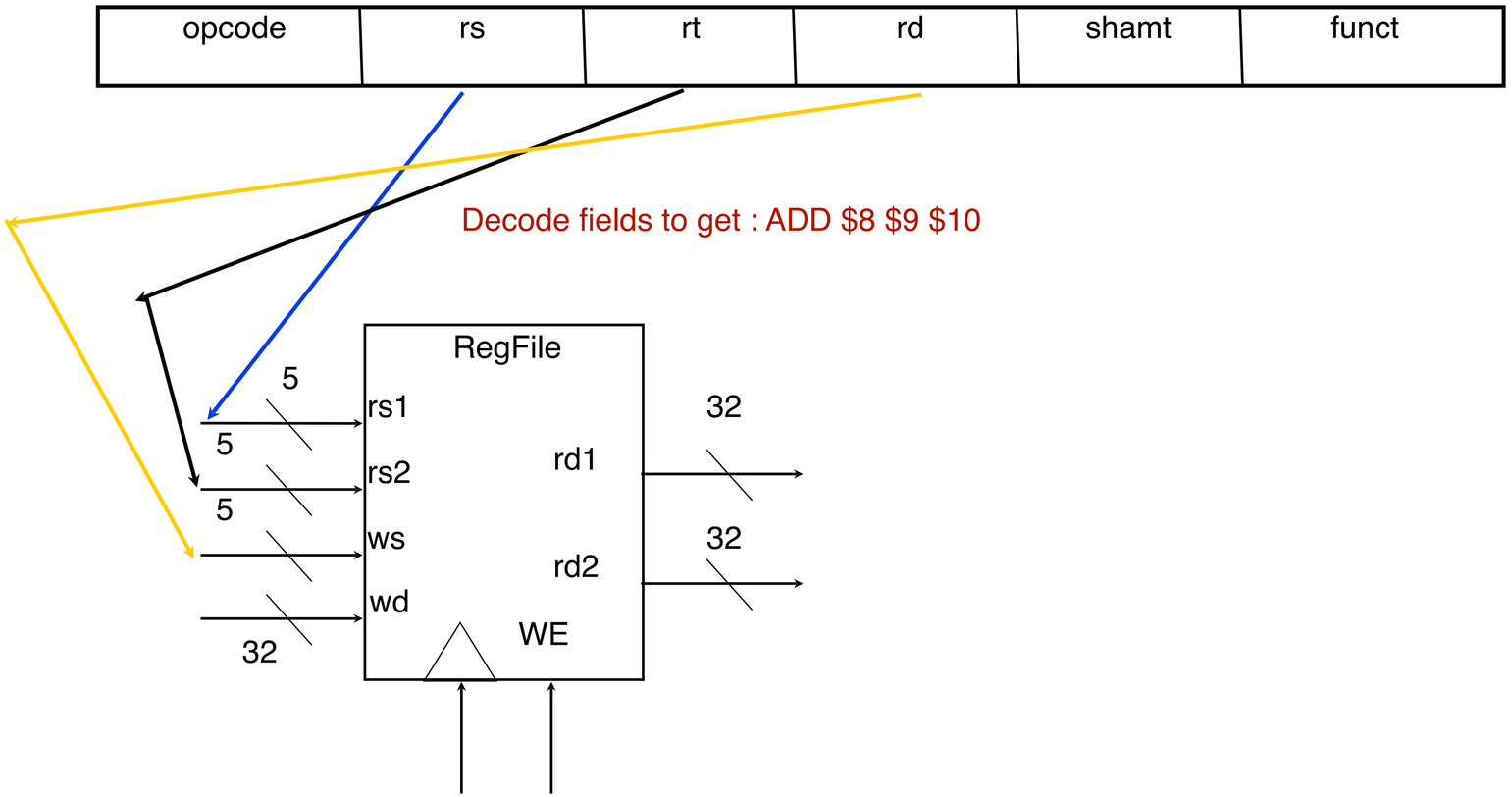



Fetching straight-line MIPS instructions requires a machine that generates this timing diagram:

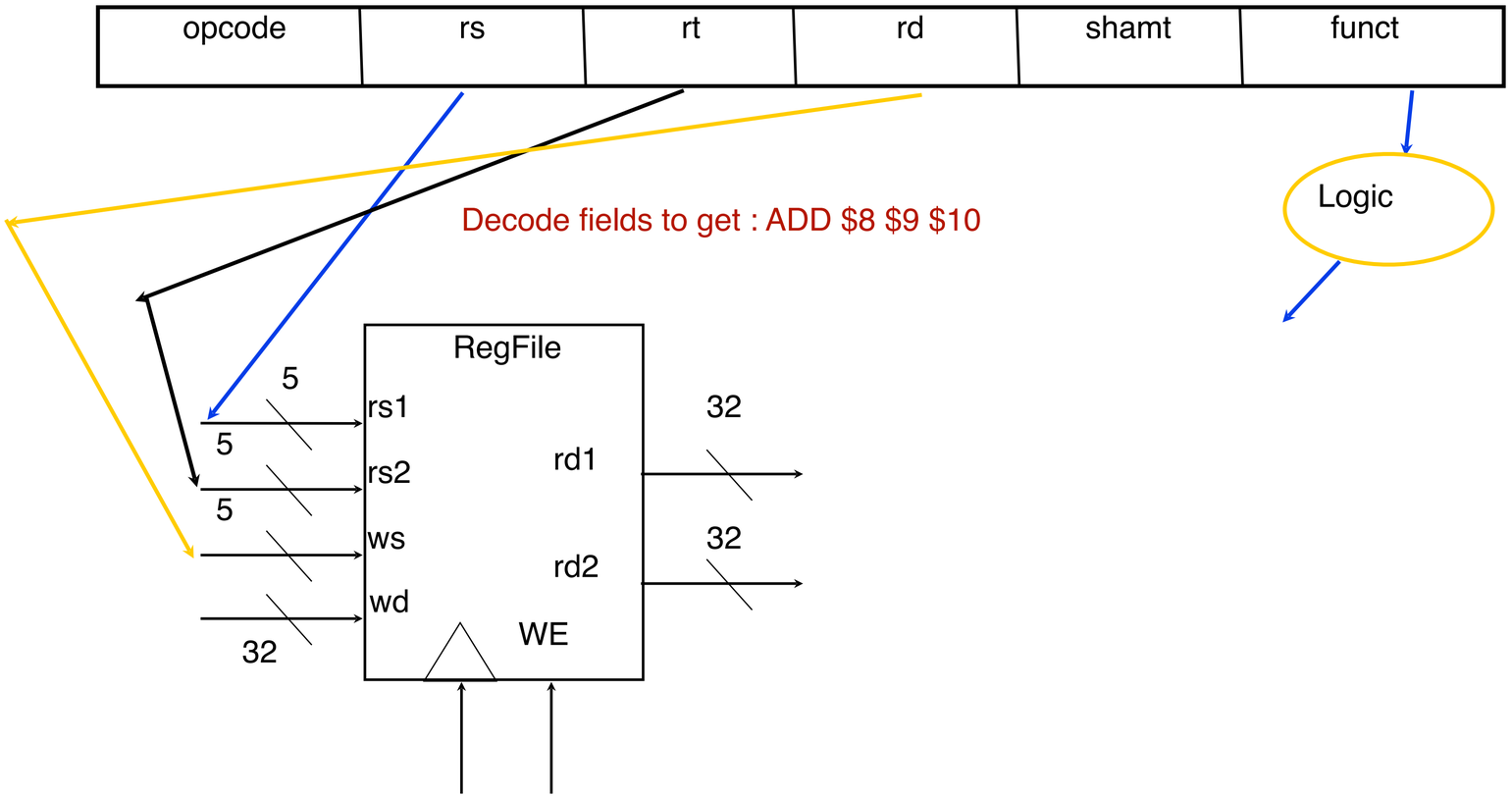
- Every clock cycle we process one instruction (super simple)
- Sending the PC and getting the instruction from memory is called **Instruction Fetch**

PC == Program Counter, points to next instruction.

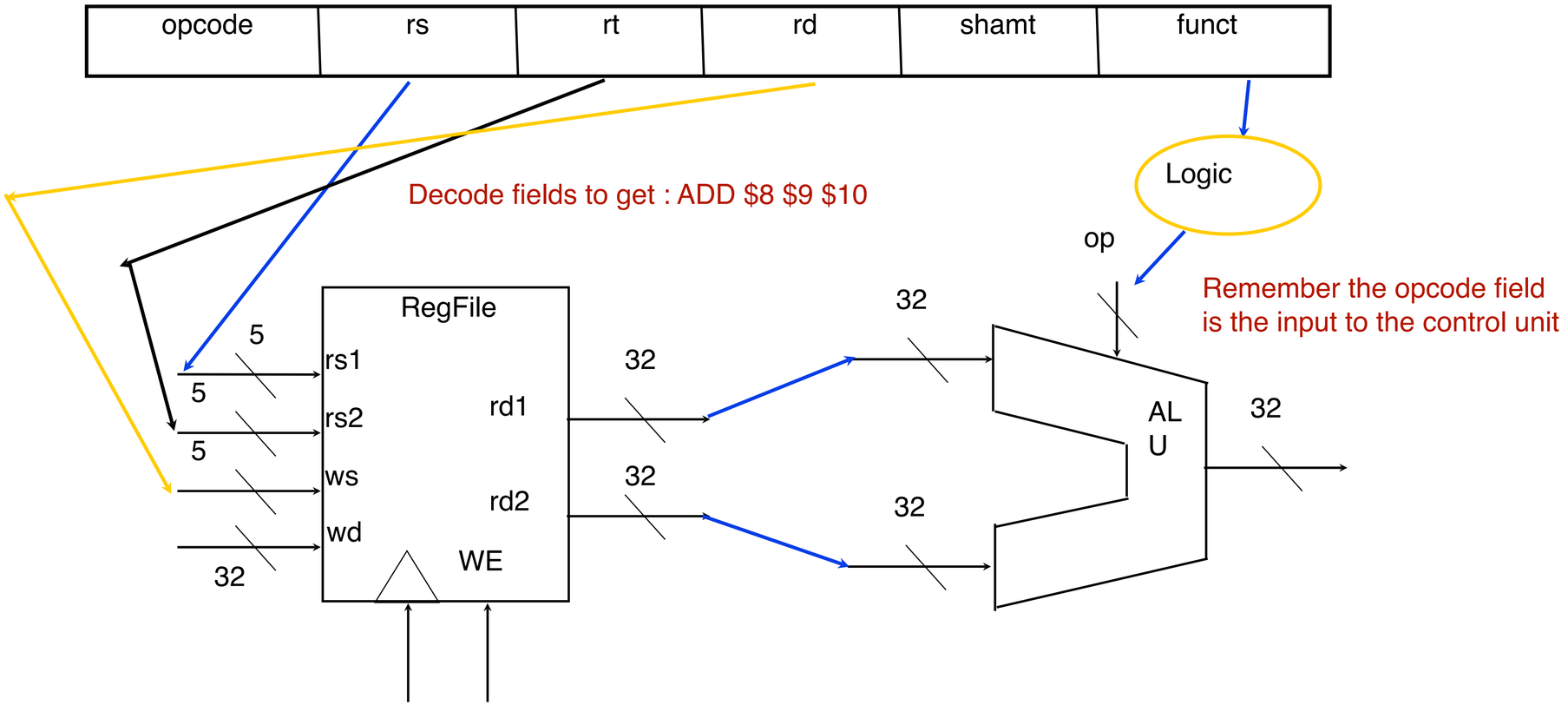
# Datapath Elements: Decoding Instructions



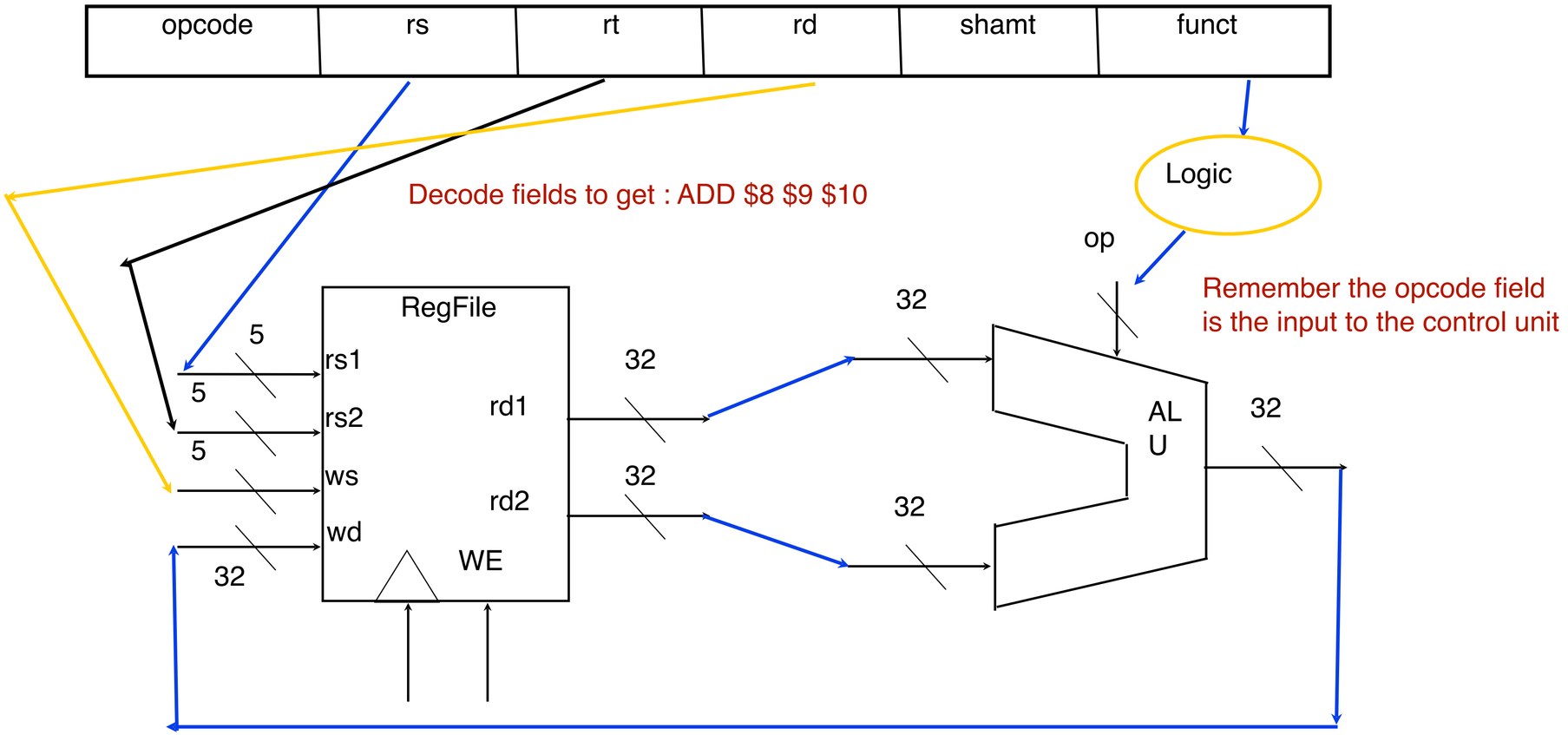
# Datapath Elements: Decoding Instructions



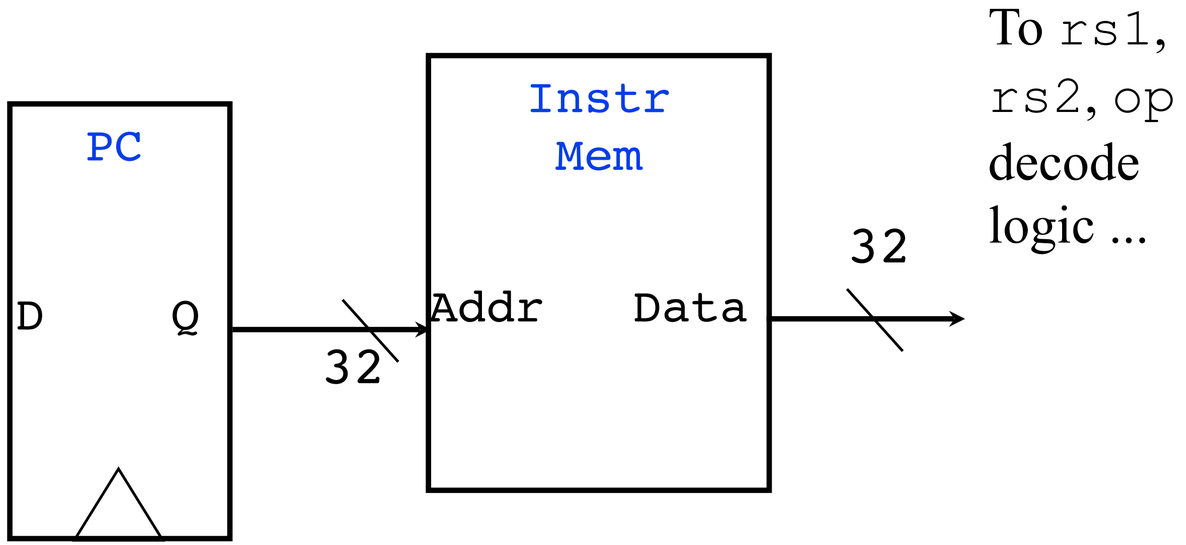
# Datapath Elements: Executing Instructions



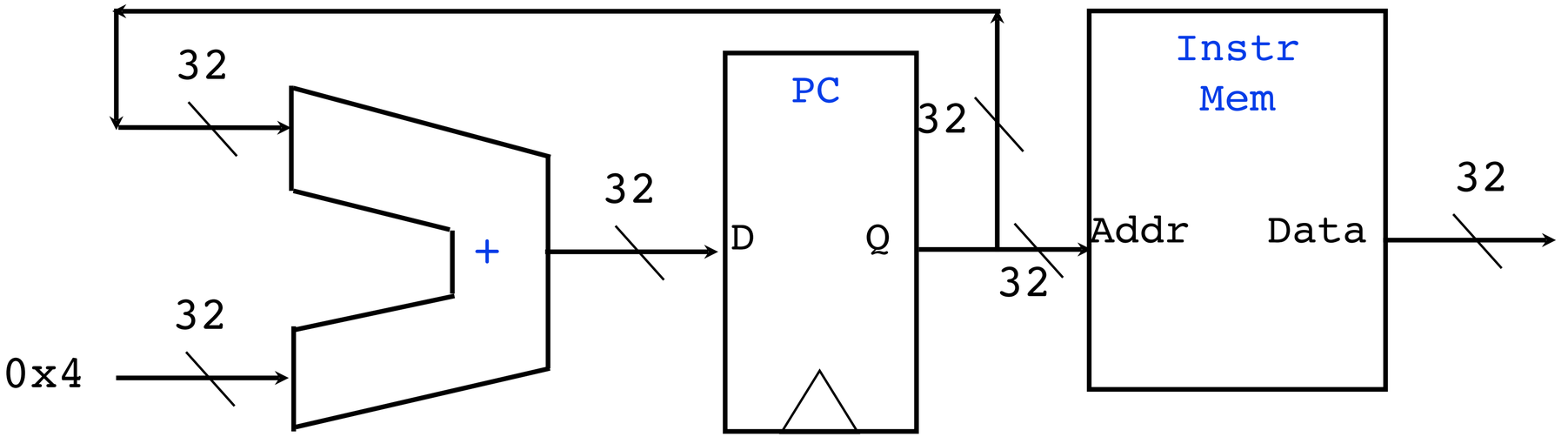
# Datapath Elements: Executing Instructions



# Datapath For Instruction Fetch

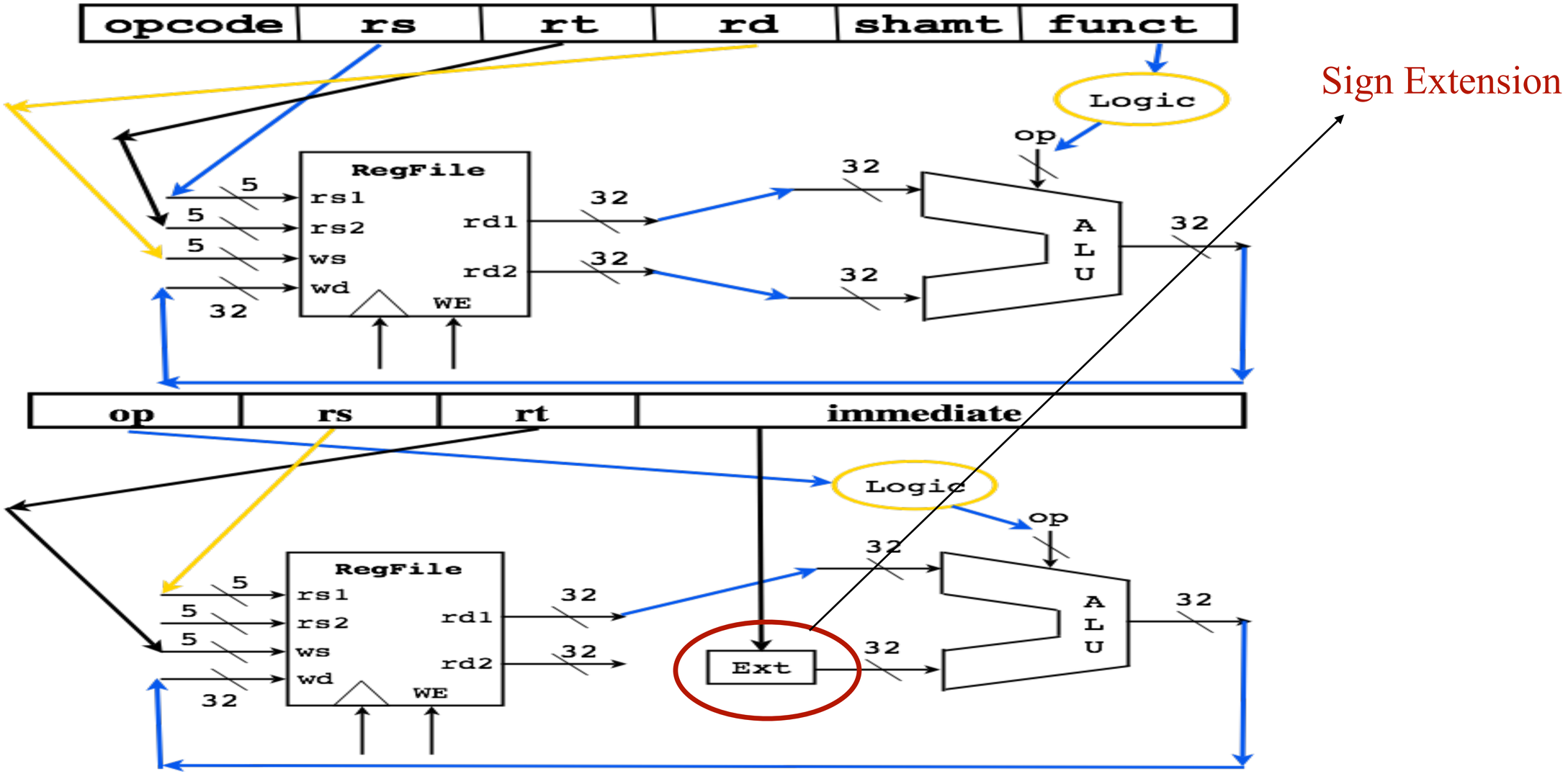


# Datapath For Instruction Fetch

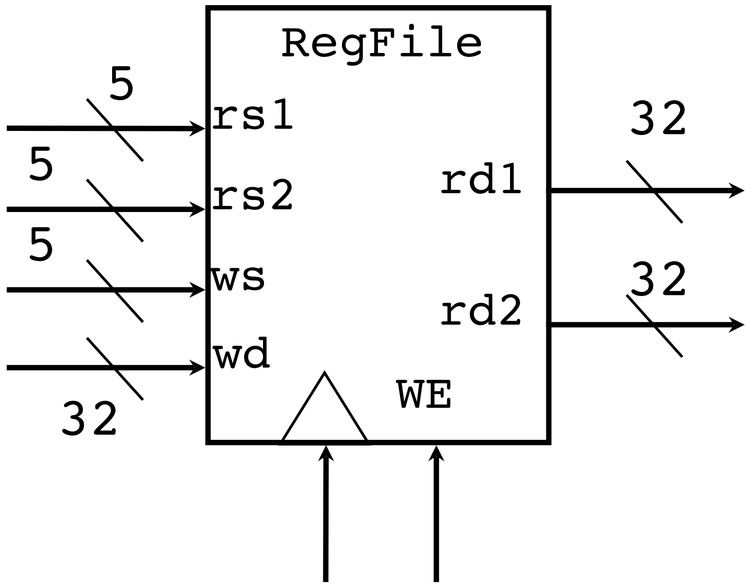


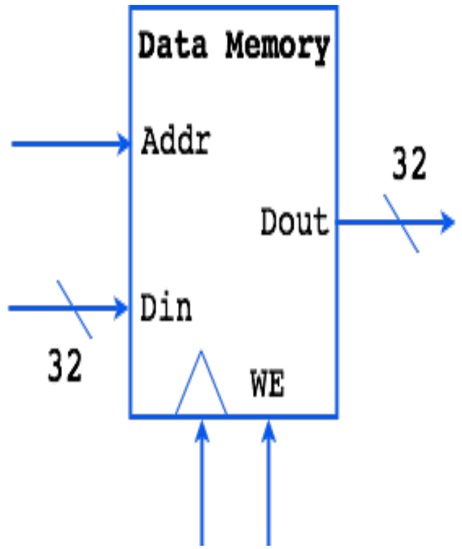

To `rs1`, `rs2`, `op` decode logic ...

# What about I format?



# Loads from Memory

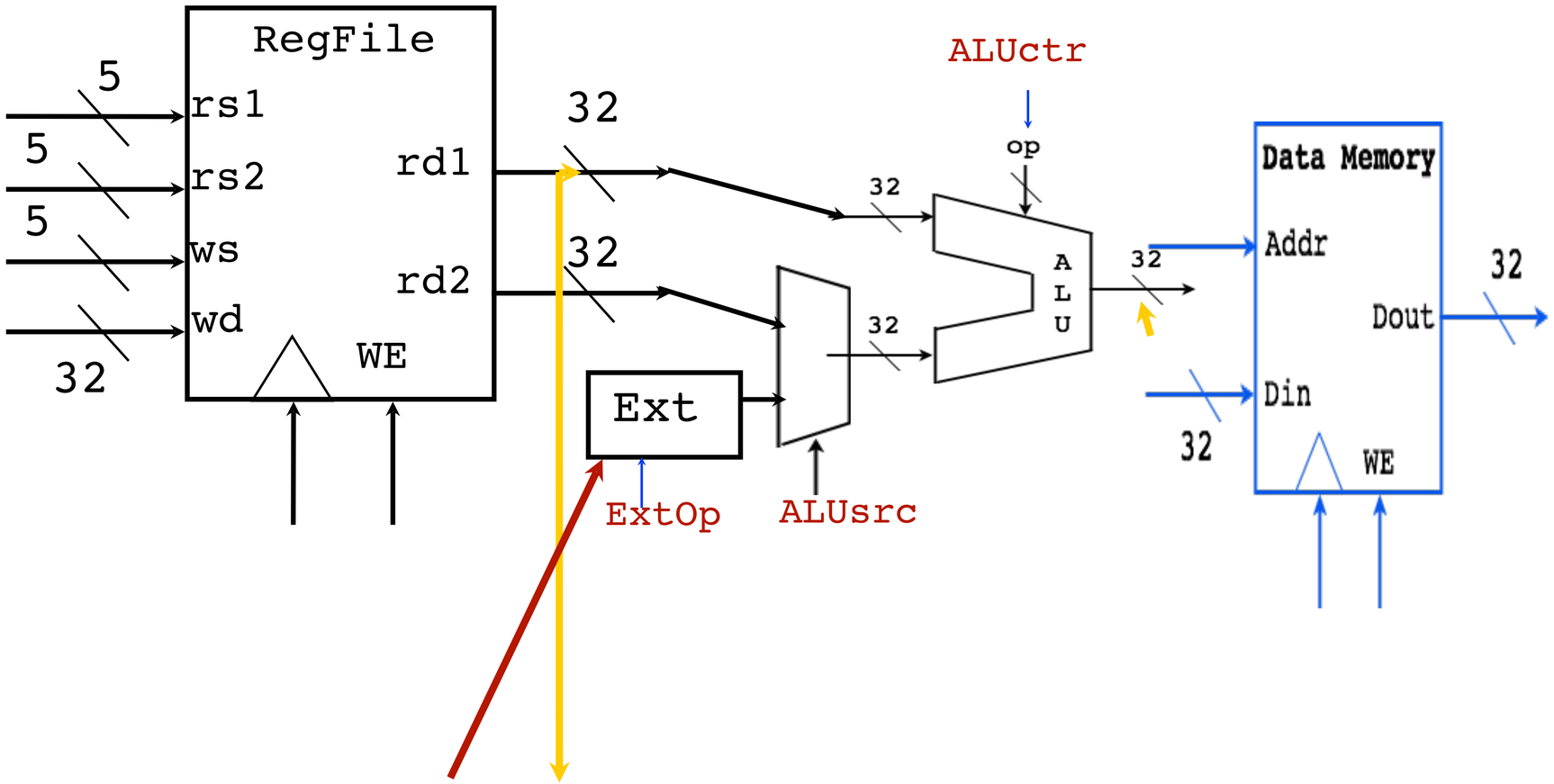



Syntax: LW $1, 32($2)

Action: $1 = M[$2 + 32]

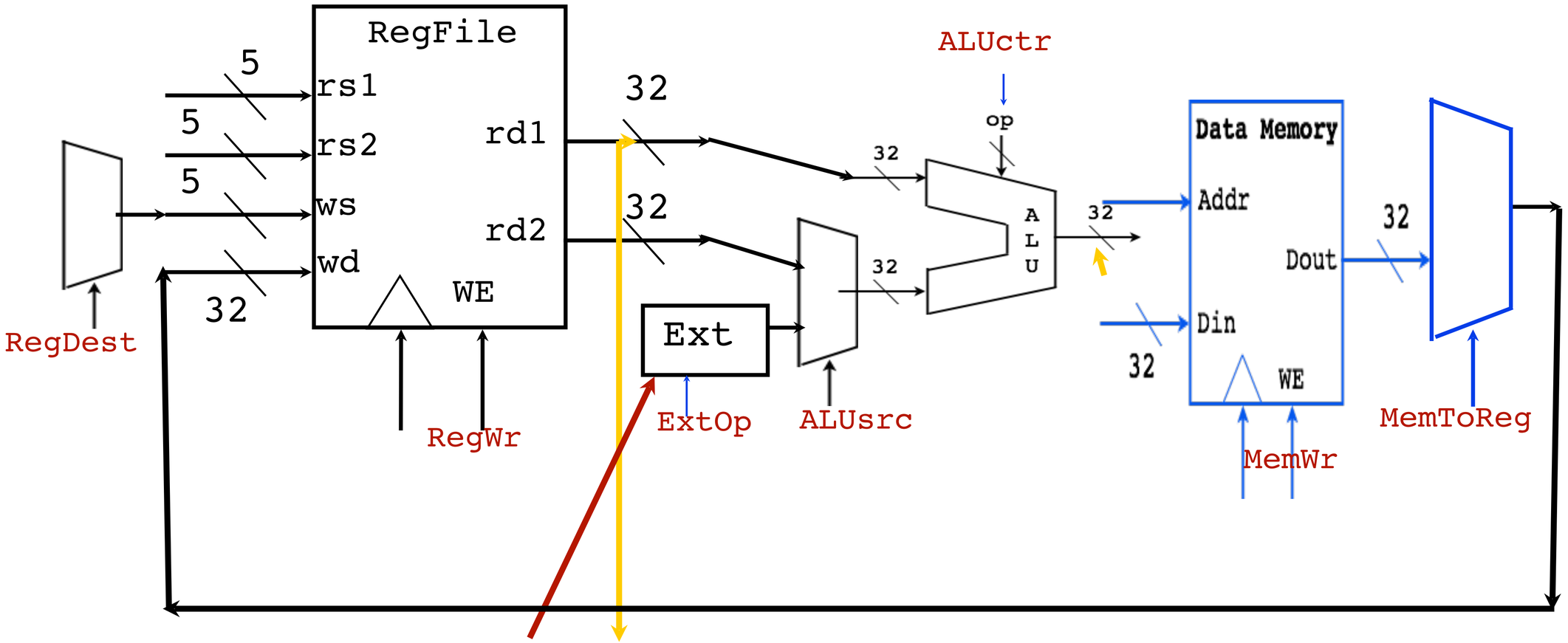
# Loads from Memory




Syntax: LW $1, 32($2)

Action: $1 = M[$2 + 32]

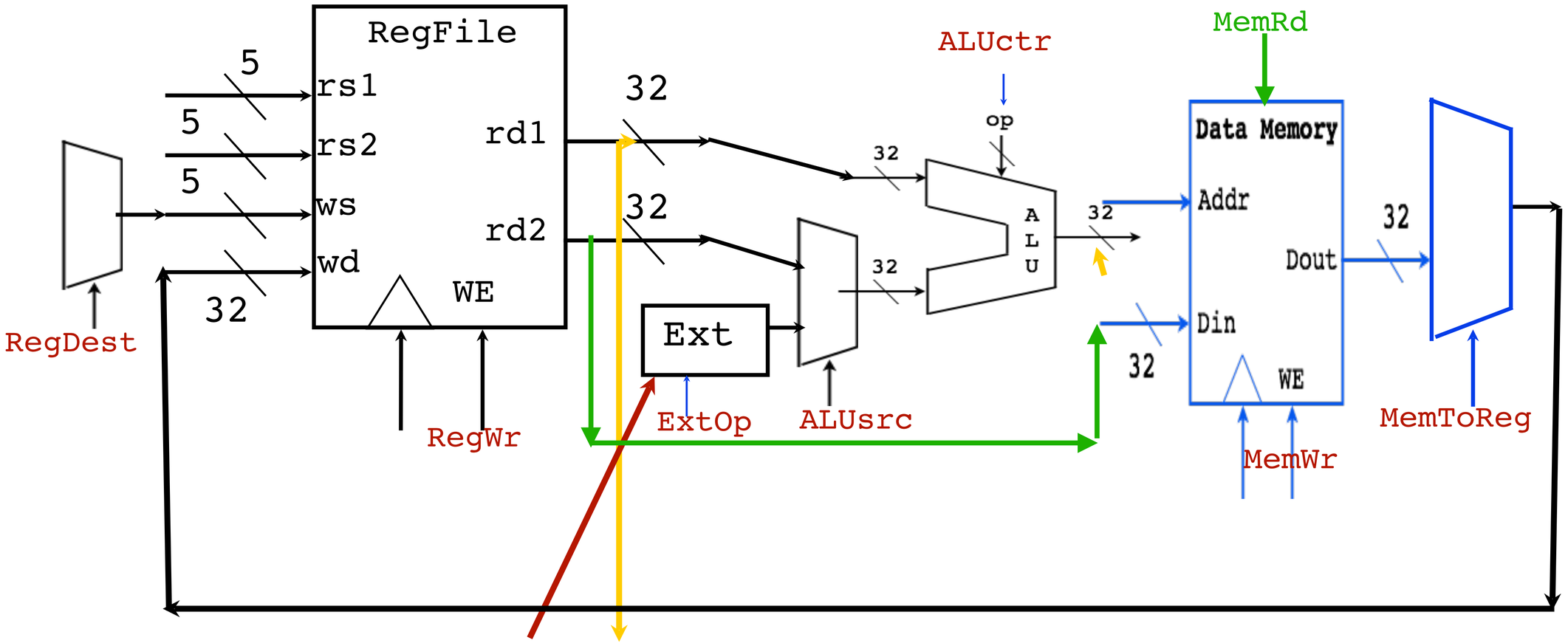
# Loads from Memory




Syntax: LW $1, 32($2)

Action: $1 = M[$2 + 32]

# Stores to Memory (with Load Datapath)

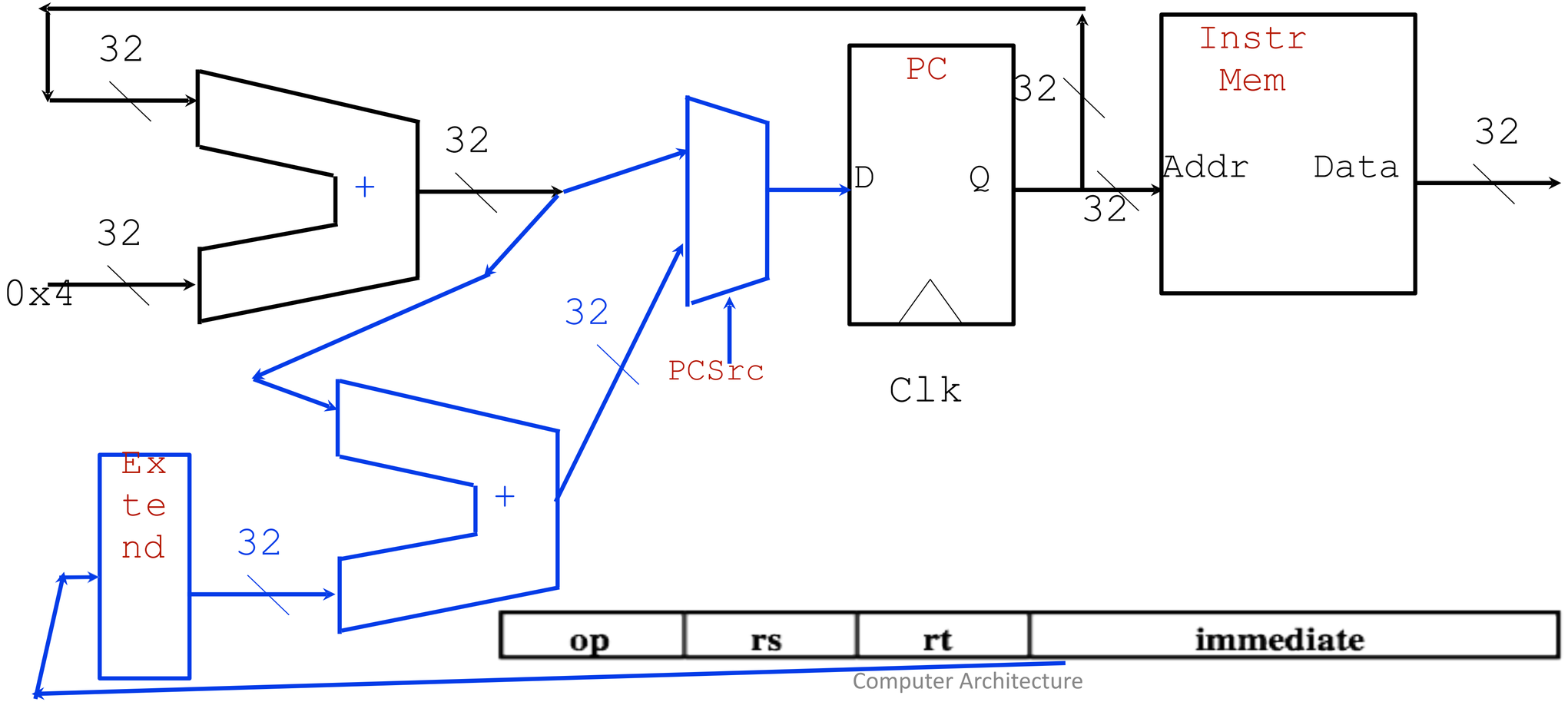



Syntax: SW $1, 32($2)

Action: M[$2 + 32] = $1

# Branch Instructions

```
Syntax: BEQ $1, $2, 12
Action: If ($1 != $2), PC = PC + 4
Action: If ($1 == $2), PC = PC + 4 + 48
```



# Branch Instructions

```
Syntax: BEQ $1, $2, 12
Action: If ($1 != $2), PC = PC + 4
Action: If ($1 == $2), PC = PC + 4 + 48
```